IC-Cloud: Computation Offloading to an Intermittently-Connected Cloud

Cong Shi, Pranesh Pandurangan, Kangqi Ni, Juyuan Yang,
Mostafa Ammar, Mayur Naik, Ellen Zegura
School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA
{cshi7, ppandura, kni3, jyang327, ammar, naik, ewz}@cc.gatech.edu

ABSTRACT

Offloading computation-intensive components of mobile applications to the cloud is of great potential to speedup the execution and reduce the energy consumption for mobile devices. The gain from computation offloading is typically counterbalanced by communication costs and delays. It is, therefore, important to undertake offloading decisions based on future prediction of Internet access timeliness and quality. Previous approaches have considered this question under the assumption that network connectivity is relatively stable. In this paper, we present IC-Cloud, a computation-offloading system for mobile environments where Internet access to remote computation resources is of highly variable quality and often intermittent. IC-Cloud uses three key ideas: lightweight connectivity prediction, lightweight execution prediction and prediction use in a risk controlled manner to make offloading decisions. Our connectivity-prediction method only uses the signal strength and user historical information to obtain a coarse-grained estimation of the network access quality. Our execution-prediction mechanism uses machine learning on dynamic program features to automatically, efficiently, and accurately predict the execution time of offloadable tasks, both on the phone and in the cloud. Acknowledging the uncertainties in these predictions, we propose a risk-control mechanism to reduce the impact of inaccurate predictions. We implemented IC-Cloud on Android and tested the system with different applications in various types of mobile environment. Results we obtained from the prototype show speedup and energy consumption reduction benefits in many computational contexts and intermittent connectivity environments.

1. INTRODUCTION

The idea of offloading computation from mobile devices to remote servers to improve performance and reduce energy consumption has been around for more than a decade [3, 4]. Its usefulness hinges on the ability to achieve computation speedups with small communication cost. In recent years, this idea has received more attention because of the significant rise in the sophistication of mobile computing applications and the availability of improved connectivity options for mobile devices. Some commercial applications such as Siri [2] have had the goal to provide sophisticated services to mobile users pervasively. Through dynamically identifying the offloadable tasks at runtime, recent work [11, 10, 17, 22] has aimed to generalize this approach to benefit more mobile applications without the burden of offloading logic.

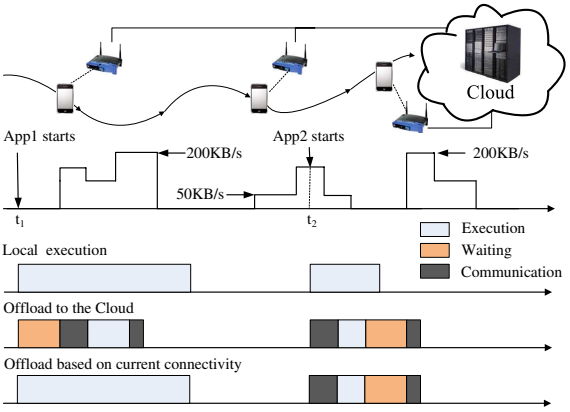


Figure 1: Simple examples showing the impact of intermittent connectivity on computation offloading.

Complicating the offloading function today is the fact that mobile users typically experience intermittent connectivity to the Internet and highly variable access quality even when connectivity exists. According to recent studies [6, 12, 23], 3G access is only available 87% of the time even in a metropolis, while WiFi coverage is even more intermittent. Figure 1 shows an example scenario where a mobile device is experiencing variable and intermittent connectivity. The uncertainties in connectivity make computation offloading challenging in two ways. First, it is hard to accurately estimate the communication cost and computation time, both of which are needed to make the offloading decision. Second, it requires computation-offloading systems to properly handle the uncertainty to avoid degrading performance.

Previous systems for computation offloading have often assumed stable network connectivity making them perform poorly when connectivity characteristics are variable and uncertain. Figure 1 provides some examples demonstrating how computation-offloading systems that do not explicitly handle intermittent or variable connectivity may degrade the application's performance. A mobile user connects to the cloud with varying access quality from time to time. She

starts two computation-intensive applications at t_1 and t_2, respectively. Consider three simple strategies for computation offloading, i.e., *Local-execution*, *Offload-to-cloud* and *Offload-based-on-current-connectivity*. For App1, *Offload-to-cloud* achieves the best performance because App1 has long local execution time and, thus, may benefit from waiting for future connectivity to offload computation. Meanwhile, for App2, *Local-execution* achieves the best performance because the mobile device loses connectivity before receiving the results. *Offload-based-on-current-connectivity* has the worst performance for both applications. It should be noted that none of these simple strategies are able to always achieve good performance. Thus a robust solution must be able to adapt the strategy.

In this paper, we propose IC-Cloud, a computation-offloading system that is designed to handle all the above-mentioned challenges in the mobile environment. To achieve this, IC-Cloud uses three key techniques: lightweight connectivity prediction, lightweight execution prediction and prediction use in a risk-controlled manner to make offloading decisions. Our connectivity-prediction algorithm only uses the signal strength and user historical information to obtain a coarse-grained estimation of the network access quality. Our execution-prediction mechanism uses machine learning on program features to automatically, efficiently, and accurately predict the execution time of offloadable tasks. Acknowledging the uncertainties in these predictions, we propose a risk-control algorithm to reduce the impact of inaccurate predictions.

We have implemented a prototype of IC-Cloud on Android and tested the system using a Samsung Galaxy Tab equiped with both WiFi and 3G and an 8-core server for offloading. We modified three applications (i.e., face detection, voice recognition and chess) to use IC-Cloud for offloading. We conducted extensive experiments in three different mobile environment. In all these experiments, IC-Cloud helps improve the performance of mobile applications and reduce the energy consumption. It achieves 4.1x speedup and reduces energy consumption to 22% in some scenarios.

The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 presents the overview of IC-Cloud's architecture. The design details of connectivity prediction, execution prediction, and computation offloading are described in Section 4, 5 and 6, respectively. The evaluation of IC-Cloud is provided in Section 7. Section 8 concludes the paper.

2. RELATED WORK

The concept of cyber foraging [27], i.e., dynamically augmenting mobile devices with resource-rich infrastructure, was proposed more than a decade ago. Since then signifcant work has been done to augment the capacity of resource-constrained mobile devices using computation offloading [3, 4, 18, 5, 31, 25]. A related technique for remote processing of mobile applications proposes the use of cloudlets which provide software instantiated in realtime on nearby computing resources using virtual machine technology [28].

Closer to our work, MAUI [11] enabled mobile applications to improve their performance and reduce the energy consumption through automated offloading. To profile the communication cost and computation gain, MAUI periodically measures the network bandwidth and uses the previous invocations to profile applications. Similarly, CloneCloud [10] can minimize either energy consumption or execution time of mobile applications through automatically identifying computation intensive methods and offloading those methods that achieve best performance. ThinkAir [22] enables scalable offloading of multiple applications with server-side support. All of these systems assume a stable environment where network connectivity and application execution time are easy to predict. In contrast, IC-Cloud targets the more challenging mobile environment where the Internet access is of highly variable quality and often intermittent.

The challenges of computation offloading in such mobile environments have been identified in [29]. A system, Serendipity [30], was also designed for computation offloading among intermittently connected mobile devices. However, to our best knowledge, IC-Cloud is the first system for computation offloading to intermittently connected cloud.

Other works, like COMET [17], enable the offloading of multi-threaded applications using distributed shared memory. ECOS [16] focuses on the data privacy in computation offloading. A detailed survey of cyber foraging can be found in [14].

Our work is related to the efforts at predicting network connectivity. BreadCrumbs [24] predicts the future locations of a mobile user by tracking her movements and creating a predictive model based on the observed data. Combined with a database of network connectivity of those locations, BreadCrumbs is able to predict future network connectivity. Deshpande et al. [13] proposed to predict the connectivity to WiFi from vehicles and use this information to improve vehicular WiFi access. These methods use energy-consuming GPS to obtain the location information. They are not suitable for energy-constrained mobile devices. Instead, IC-Cloud uses an energy-efficient method to predict the connectivity properties that are critical to computation offloading.

Our work is also related to the studies on the prediction of program-execution time. Gupta et al. [19] used a variant of decision trees to predict execution-time ranges for database queries. Ganapathi et al. [15] used KCCA to predict time and resource consumption for database queries. These approaches either require manual efforts to identify good features, or require applications to high correlations between input size and execution time. Mantis [9] estimated the execution time of an entire application using executable program slices to obtain feature values at runtime. In contrast, IC-Cloud estimates the exeuction time of each offloadable function.

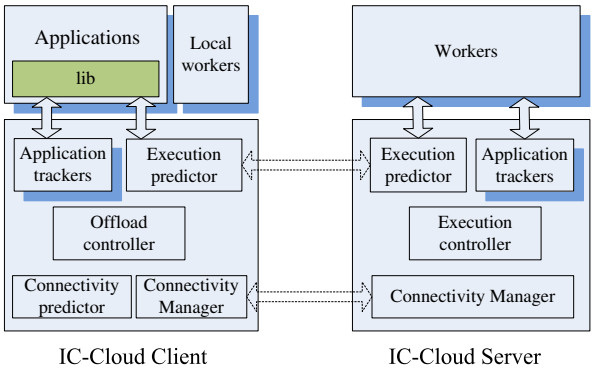


Figure 2: Overview of IC-Cloud system architecture.

3. IC-CLOUD ARCHITECTURE

IC-Cloud aims to achieve effective computation offloading in a mobile environment where Internet access to remote computation resources is of variable quality and even intermittent. Figure 2 shows the high-level architecture of IC-Cloud. On the mobile device, IC-Cloud consists of six major components: 1) a *connectivity predictor* that monitors the network states and maintains a database of the historical information of the network states; 2) a *connectivity manager* that handles data transfer between the mobile device and the cloud; 3) an *execution predictor* that collects program features from the applications and predicts the execution time of tasks considered for offloading; 4) a set of *application trackers* each of which monitors the offloaded tasks for an application and adjusts its strategy based on the connectivity over time; 5) an *offloading controller* that uses the information from the connectivity predictor and the execution predictor to decide if a task should be offloaded to the cloud; 6) *local workers* that may execute some offloaded tasks in the case that they cannot return the results in time.

On the server, IC-Cloud consists of four major components: 1) an *execution predictor* that tracks the program features of the applications and sends them back to the mobile device; 2) a set of *application trackers* that monitor the execution of offloaded tasks; 3) an *execution controller* that controls the execution of the offloaded tasks; 4) a *connectivity manager* that communicates with the mobile devices.

Connectivity prediction is critical to the performance of IC-Cloud as the offloading decision relies on the prediction of future connectivity. There are two major concerns in its design: accuracy and energy-efficiency. Our main idea is to use the signal strength and the user's historical information to predict connectivity. Many studies [24, 13] have shown that it is sometimes possible to predict user mobility and, thus, their connectivity using the user's historical information. However, many of these prediction mechanisms are energy-consuming as they require GPS location to achieve accurate prediction. Instead of trying to obtain accurate connectivity prediction, IC-Cloud only uses the perceived signal strength to achieve coarse-grained prediction of the connectivity in an energy-efficient way and lets the offloading controller handle the uncertainties in the prediction. We provide further details in Section 4.

Similar to MAUI [11] and ThinkAir [22], IC-Cloud provides a library to application developers and allows them to annotate all the tasks to be considered for offloading. IC-Cloud will also instrument these applications to collect features for all these offloadable tasks. Then IC-Cloud uses machine learning on these features to accurately predict their execution time online. The detailed design of the execution predictor is presented in Section 5.

Using the information from connectivity prediction and execution prediction, the offloading controller estimates the potential benefits to offload the computation. Due to the inherent uncertainties in user mobility and the connectivity prediction, the offloading controller will sometimes make wrong decisions. Therefore, it is essential to take the risk into account. In addition, different applications may tolerate different risks. For example, interactive applications (e.g., games) should be executed before their deadlines and are less tolerant to extra delays, while some background applications (e.g., virus scanning) can tolerate occasional extra delays. Therefore, IC-Cloud should allow applications to specify their tolerance to risks. The detailed design of the offloading controller is described in Section 6.

4. CONNECTIVITY PREDICTION

A fundamental challenge to the design of IC-Cloud is how to estimate the communication cost in the mobile environment. When offloading a piece of computation to the cloud, the communication cost consists of the time to transfer the data from the mobile device to the cloud and the time to return the result to the mobile device after execution. When the connectivity is intermittent, the cost may also include the time to wait for connectivity in either direction, making the estimation even more complicated.

A heuristic solution to this problem is to predict the Internet access quality and then use the predicted values to assess the communication cost. Developing a highly accurate prediction method will be beneficial to IC-Cloud but is out of the scope of this paper. We expect a simple and energy-efficient method to have advantages over greater accuracy but also greater energy, which predicts future connectivity with user historical information. IC-Cloud maintains a database of the perceived signal strengths and the achieved throughput when using the network. Such information is easy to obtain and requires little energy to collect. This prediction method can be replaced by other more accurate methods. Our focus in this paper is on profiling the connectivity and assessing the communication cost for computation offloading.

To illustrate how dynamic Internet access quality impacts the communication cost of computation offloading, we plot the measured WiFi signal strength on a Georgia Tech cam-

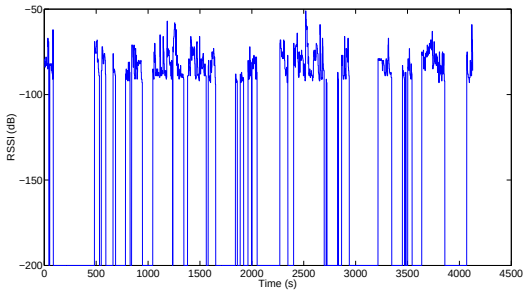


Figure 3: The WiFi signal strength measured on a campus shuttle. When the mobile device disconnects from WiFi, the signal strength is set to -200.

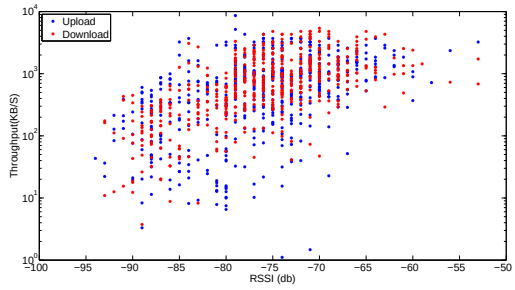


Figure 4: Measured throughput vs. WiFi signal strength.

pus shuttle in Figure 3 and the throughput to a server in our lab in Figure 4. The figures demonstrate three key properties that impact the estimation of communication cost: intermittent connectivity, varying signal strength over time and uncertain throughput given the signal strength. The latter two properties are also very common for WiFi in the indoor environment and cellular 3G data access. We describe how to profile each of these three properties in the following three subsections.

4.1 Intermittent Connectivity

Intermittent connectivity primarily impacts the estimation of communication cost in the following two ways. First, if the mobile device disconnects from the server when a task is to be offloaded, IC-Cloud needs to decide if it should wait for the next connectivity to offload the task. If the local execution time is very long while the next connectivity will come soon, it may be beneficial to wait. Otherwise, it should start to execute the task immediately on the mobile device. Second, it is also possible that the mobile device will lose connectivity to the server after offloading the task. Therefore, before offloading the task IC-Cloud should estimate the possibility of disconnection before obtaining the result from the server. In addition, if it loses connectivity in the middle of offloading, it also needs to decide whether to wait for the result or restart a local worker for the task. To assess the communication cost under intermittent connectivity, we need to estimate the residual duration of current connectivity, if connected, or the start time of the next connectivity, if disconnected.

Let us use C to denote the duration of a contact during which the mobile device can always communicate with the server and use D to denote the duration of an inter-contact during which the mobile device totally loses connectivity to the server. Let us also use C_t to represent the duration of current contact until time t if it is connected at time t and use D_t to represent the duration of the inter-contact until time t. Finally, let $R_{C,t}$ and $R_{D,t}$ denote the residual duration of a contact and that of a inter-contact since time t, respectively. Therefore, the expected values and the standard deviations of the residual durations can be computed as follows:

$$E(R_{X,t}) = \int_{X_t}^{+\infty} \frac{(X - X_t)f(X)}{P(X \geq X_t)} dX \tag{1}$$

$$\sigma(R_{X,t}) = \sqrt{\int_{X_t}^{+\infty} \frac{(X - X_t)^2 f(X)}{P(X \geq X_t)} dX} \tag{2}$$

where X is either C or D.

There are also the cases where C_t or D_t are not available. For example, the user just restarted the mobile device. Let's consider the scenario that it is within a contact at time t. The possibility that it is within a contact of duration C is proportional to its duration, i.e., $\frac{Cf(C)}{E(C)}$. Since $R_{C,t}$ can be any value within $[0, C]$ under the condition that the contact duration is C, its expected value is $\frac{C}{2}$. Thus, the expected value and the standard deviation of $R_{C,t}$ are:

$$E(R_{C,t}) = E(\frac{Cf(C)}{E(C)} \times \frac{C}{2}) = \frac{E(C^2)}{2E(C)} \tag{3}$$

$$\begin{aligned} \sigma(R_{C,t}) &= \sqrt{\int_0^{+\infty} \frac{Cf(C)}{E(C)} \int_0^C \frac{(x - \frac{C}{2})^2}{C} dx dC} \\ &= \sqrt{\frac{E(C^3)}{12E(C)}} \end{aligned} \tag{4}$$

The results for $R_{D,t}$ are similar to the above two equations, i.e., replacing C with D in these equations.

4.2 Varying Signal Strength

IC-Cloud uses the signal strength as an indicator of Internet access quality because the wireless interface is usually the bottleneck of network performance in mobile environment. The communication cost of computation offloading includes both the time of sending data to the cloud and that of receiving the result from the cloud. The former uses the current signal strength for estimation, while the latter requires an estimate of the distribution of future signal strength and uses it for cost estimation. In this subsection, we describe how to obtain the distribution of future signal strength based on user historical information.

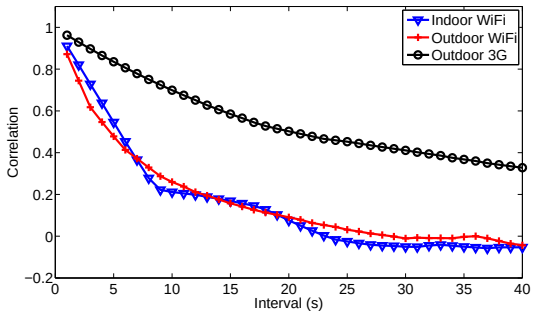


Figure 5: The correlation between current WiFi signal strength with future signal strength. The X-axis is the time difference.

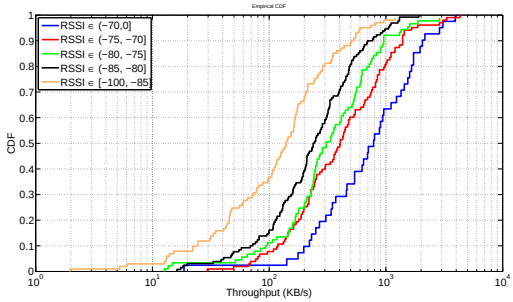


Figure 6: The distributions of the measured WiFi throughput on a Georgia Tech campus shuttle. They are divided into five categories based on the signal strength.

Our method is to use the current signal strength and the statistical distribution of user historical information to obtain that distribution. To demonstrate how current signal strength can be used to obtain the distribution of future signal strength, we plot the correlation between the signal strength at time t and that at time $t + \Delta t$ for three different mobile environments in Figure 5. "Indoor WiFi" corresponds to the WiFi measurement conducted by a student randomly walking in a building with good WiFi coverage; "Outdoor WiFi" refers to WiFi measurement conducted on a Georgia Tech campus shuttle, the same as Figure 3; "Outdoor 3G" refers to 3G measurement conducted by a student on the commute between home and school. When Δt is small, the correlations in all these scenarios are high. However, the correlation of WiFi signal strength (both indoor and outdoor) quickly drops from about 0.9 to about 0.2 when Δt increases from 1 second to 10 seconds. Meanwhile the correlation of 3G signal strength is still about 0.4 when $\Delta t = 30$. If the correlation is larger than a threshold, we can obtain the distribution of future signal strength using the current signal strength.

Let us use $\langle x(t), x(t + \Delta t)\rangle$ to denote the pair of signal strengths at time t and $t + \Delta t$. We simply assume that $x(t)$ follows the normal distribution $N(u, \sigma^2)$ and use $\rho_{\Delta t}$ to denote the correlation between $x(t)$ and $x(t + \Delta t)$. In the implementation, u, σ^2 and $\rho_{\Delta t}$ are obtained from user historical information. Then, $\langle x(t), x(t + \Delta t)\rangle$ follows a bivariate normal distribution:

$$N\left(\begin{pmatrix} u \\ u \end{pmatrix}, \begin{pmatrix} \sigma^2 & \rho_{\Delta t}\sigma^2 \\ \rho_{\Delta t}\sigma^2 & \sigma^2 \end{pmatrix}\right)$$

Using the conditional distribution of bivariate normal distribution [8], we can obtain the distribution of $x(t + \Delta t)$ given $x(t)$ as:

$$N(u + \rho_{\Delta t}(x(t) - u), (1 - \rho_{\Delta t}^2)\sigma^2) \quad (5)$$

It is noteworthy that when $\rho_{\Delta t}$ is small, the variance, $(1 - \rho_{\Delta t}^2)\sigma^2$, will be large, indicating inaccurate estimation. In addition, the value of $\rho_{\Delta t}$ will also be biased in the implementation because we use the historical information to approximate it. Therefore, in our implementation, when $\rho_{\Delta t}$ is smaller than a threshold (e.g., 0.4), IC-Cloud simply uses the overall statistical distribution of signal strength, namely, $N(u, \sigma^2)$, to describe the distribution.

4.3 Uncertain Throughput

To estimate the time of sending the data and receiving the result, IC-Cloud needs to predict its current throughput (for sending) as well as the future one (for receiving). There are three challenges in the prediction. First, since IC-Cloud uses the signal strength as the indicator of Internet access quality, it is hard to predict the throughput accurately. As shown in Figure 4, for any specific value of the signal strength, the measured throughput usually has high variance. Second, IC-Cloud uses historical information to obtain the relation between signal strength and throughput. It may be biased, especially when the data is sparse. Third, as discussed in the above subsection, we can only obtain a range of future signal strength, making it even harder to estimate the corresponding throughput.

To handle the uncertainties and make robust estimation, we divide the signal strength into several categories and generate a throughput distribution for each category using the historical information. Figure 6 shows an example using the measurement on the Georgia Tech campus shuttle. The throughput distributions of the various categories are quite different.

5. EXECUTION PREDICTION

Cloud execution is typically more powerful than the mobile device but it incurs the cost of offloading a task and reintegrating its result. Thus, even assuming stable connectivity, it may be beneficial to offload a task to the cloud only on certain inputs to the task, and execute it locally on other inputs. To make this decision, IC-Cloud uses a system described in this section to predict the execution time of tasks in Android apps on given inputs, both locally and in the cloud.

We identified the following four goals that the prediction mechanism must satisfy to be effective:

- *Automated*: it must be as automated as possible to re-

duce the programmer burden.

- *Efficient*: it must be reasonably lightweight at runtime to avoid outweighing the benefits of offloading.
- *Accurate*: it must be sufficiently accurate to produce correct offloading decisions.
- *General*: it must handle a large class of Android apps.

Figure 7 shows an overview of our execution prediction system designed to satisfy these criteria. The system has an offline component and an online component. The offline component runs once before deploying the app. It creates, for each designated task in the app that is offloadable, a *performance model* of the task on each computational device. We allow any dynamic instance of any function in the app to be offloaded. We found this task granularity to be sufficient in practice. Our system, however, does not infer such functions, nor the data that they input and output. It instead requires the user to specify them. Our limited experience with three real-world Android apps (Section 7) as well as our prior experience with the CloneCloud computation-offloading system [10] shows that mobile apps are relatively small programs with few tasks computation-intensive enough to be worth offloading that an app developer can manually identify without much effort. Our system instead focuses on automatically predicting the execution time of dynamic instances of such functions, once identified, under different inputs to them on each device.

The offline component synergistically combines program analysis and machine learning to build performance models, one for each specified function $\mathbf{t_1}, ..., \mathbf{t_S}$ in the app, on each computational device. These performance models are over automatically chosen app features that are the best predictors of the execution time of the functions on the devices. To obtain these features, the offline component requires the app's binary P as well as a set of training inputs $\mathbf{d_1}, ..., \mathbf{d_N}$ to the app. Besides performance models, this component also produces as output an augmented app binary Q that tracks in a lightweight style the values of features needed by the models to predict execution time on new inputs. We describe the offline component in more detail in Section 5.1.

The online component runs during the execution of app binary Q on a mobile device. It tracks the instrumented feature values and is called upon by the offload controller just before an invocation of an offloadable function (i.e., one of $\mathbf{t_1}, ..., \mathbf{t_S}$). It estimates the execution time of that invocation on both the mobile device and the cloud by applying the performance model of that function, provided by the offline component, on the current feature values. Finally, the offload controller uses these estimates along with other factors such as network connectivity, the cost of migrating data, etc. (see Section 6) in order to decide whether to execute the function on the mobile device or the cloud. We describe the online component in more detail in Section 5.2.

5.1 Offline Component

The architecture of the offline component is shown in Figure 8. Its high-level idea is to instrument the app with features that are potentially good predictors of the execution time of the specified offloadable functions (Instrumentor), then to profile the instrumented app to collect values of the features and execution times of the functions on a set of given app inputs (Profiler), and lastly to use machine learning to build a performance model based on a few features that are the best predictors (Model Generator). The Instrumentor runs only once, but the Profiler and Model Generator are run once for each computational device of interest. We next describe each of the above three steps in detail.

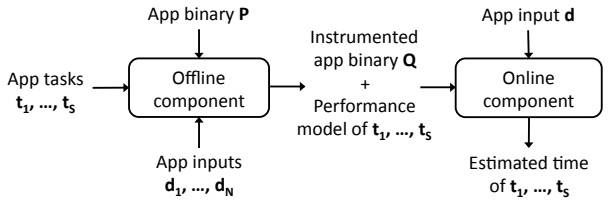


Figure 7: Overview of performance prediction system.

A feature is a good predictor if it predicts accurately and is available at the time of prediction. For example, a feature that becomes available in the middle of executing a function is useless for predicting the execution time of that function. To find features that are good predictors, the Instrumentor casts a wide net, by means of broad *feature schemes*, each of which generates a set of features of a particular kind from the app. We use the following three feature schemes:

- *Loop counts*: This scheme generates a separate feature to track the number of times each loop in an app runs.
- *Return values*: This scheme generates a separate feature to track the return value of each function call site, provided it is of integer type.
- *Parameter values*: This scheme generates a separate feature to track each parameter value of integer type of each function call site.

The loop counts and return values schemes track the cumulative value of each feature, i.e., they accumulate these values over all executions of the loop and all executions of the call site, respectively, in a particular run, whereas the parameter values scheme tracks only the most recent value of each feature. We found this combination provides complementary information and strikes a good tradeoff between tracking feature values from the recent and the distant past. All three schemes in the Instrumentor are implemented using Soot [26], a Java bytecode compiler framework, by iterating over the body of each function in each app class. Finally, the Instrumentor also injects code at the entry and exit of each of the specified offloadable functions $\mathbf{t_1}, ..., \mathbf{t_S}$, for the Profiler to measure their execution time.

The Profiler runs the instrumented app binary produced by the Instrumentor on each of given app inputs $\mathbf{d_1}, ..., \mathbf{d_N}$. Whenever an offloadable function $\mathbf{t_i}$ is about to be called, the Profiler records the values of all instrumented features $\mathbf{f_1}, ..., \mathbf{f_M}$ (say $v_1, ..., v_M$), and when the function finishes

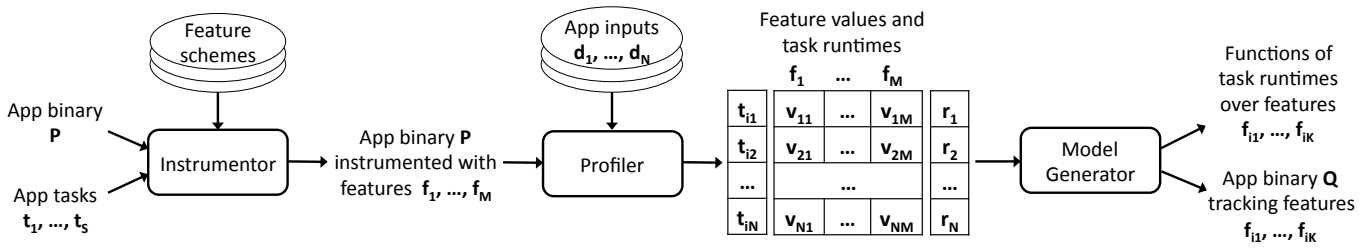


Figure 8: Architecture of offline performance model generator.

executing, it also records the execution time r of this just finished instance, and outputs tuple $\langle \mathbf{t_i}, v_1, ..., v_M, r \rangle$. Note that the same function $\mathbf{t_i}$ may be called multiple times on a single app input, providing multiple data points for an offloadable function in even a single run.

Finally, the Model Generator uses the tuples output by the Profiler, and builds a performance model separately for each offloadable function $\mathbf{t_i}$. This model is a polynomial function that approximates the execution time of $\mathbf{t_i}$ in terms of features $\mathbf{f_1}, ..., \mathbf{f_M}$. An example model is:

$$0.2 + 2.4f_3 + 0.4f_7 + 7.4f_3f_7$$

The model is inferred using the regression algorithm from [20]. Like any regression algorithm, this algorithm fits as closely as possible the feature values $v_1, ..., v_M$ and execution time r in each tuple output by the Profiler for function $\mathbf{t_i}$. But we chose this algorithm because it has two salient aspects suited for performance prediction: *sparsity* and *non-linearity*. The sparsity aspect concerns minimizing the number of features selected in the performance model (e.g., f_3 and f_7 in the above example) which is beneficial for two reasons: (i) it prevents the performance model from overfitting the training data and losing accuracy on inputs that deviate from the training inputs; and (ii) the more the number of features selected, the more heavyweight the instrumentation to track in the online component, as explained below. Finally, the non-linear aspect of the algorithm allows non-linear terms (e.g., f_3f_7) in the model, allowing to approximate execution times of real programs more accurately.

5.2 Online Component

The online component is relatively straightforward. It runs as part of the modified app binary produced by the offline component that tracks the values of features used in performance models of offloadable functions. The tracking overhead is negligible due to the sparse nature of the constructed models. Whenever an offloadable function is about to be invoked on the mobile device, the online component is called upon by the offload controller of IC-Cloud, at which point it plugs the current values of the tracked features into the perfomance models of that function for both the mobile device and the cloud, and provides the estimated execution times to the controller. The next section describes how the controller uses this information in making the offloading decision.

6. COMPUTATION OFFLOADING

The offload controller of IC-Cloud uses the information from the connectivity and execution predictors to decide how to offload the computation-intensive tasks of mobile applications. Ideally, if future connectivity and execution time can be accurately predicted immediately after the mobile application starts, a global optimal solution [29] can be used to make the offloading decision. However, such global optimum is unavailable in the mobile environment investigated in this paper because of two reasons. First, the dynamic Internet access quality makes it hard to accurately predict the communication costs for all the offloadable functions, especially for those to be called later. Second, some of the application features essential for the execution prediction of offloadable functions may be unavailable until these functions are about to be called.

Instead, the offload controller uses a greedy strategy to make the offloading decision. Every time when an offloadable function is called, the offload controller determines if it is beneficial to offload the function. Because of the uncertainty inherent in the mobile environment, the offloading decision takes risk into consideration. In case a bad decision has been made, it will also adjust its strategy with new information available. Meanwhile, for functions capable of executing concurrently, the offload controller will maximize their overall benefit. We describe these design details in the following subsections.

6.1 Offloading Gain

When an offloadable function is called at time t, the offload controller needs to determine if it is beneficial to offload this function to the cloud. Let us use T_{ws} to denote the time to wait for connectivity before sending the data, T_s for the time to send the data, T_c for the execution time in the cloud, T_{wr} for the time to wait for connectivity before receiving the result, T_r for the time to receive the result, and T_l for the local execution time on the mobile device. Let G represent the gain of offloading the function. Therefore,

$$G = T_{ws} + T_s + T_c + T_{wr} + T_r - T_l. \quad (6)$$

In Formula 6, T_c and T_l are independent of network connectivity and can be estimated using the techniques described in Section 5. Meanwhile, T_{ws}, T_s, T_{wr} and T_r can be estimated using the information from the connectivity predictor.

The detailed formulas are described in the appendix.

Conceptually, when $G > 0$, it will be beneficial to offload the function. However, because of the uncertainties in the mobile environment, the offload controller can only obtain a distribution for G (i.e., $E(G)$ and $\sigma^2(G)$). Simply using $E(G)$ to make the offloading decision will introduce the risk of longer execution time and, thus, cause bad user experience. Therefore, controlling the risk in offloading is very important. We describe the risk-control mechanism of the offload controller in the next subsection.

6.2 Risk Control

Our risk-control mechanism consists of two major parts. First, when making the offloading decision, the offload controller should consider the risk as well as its gain. Second, in the case that a bad decision is made, the offload controller will adjust the strategy with new information available.

In the first part, the offload controller has two options with different returns and risks. If it decides to execute locally, both return and risk are 0. If it decides to offload the function, its return and risk are $E(G)$ and $\sigma(G)$, respectively. Instead of only using the return (i.e., $E(G)$), the offload controller uses the risk-adjusted return [7] (i.e., $\frac{E(G)}{\sigma(G)}$) to make the offload decision. IC-Cloud will offload a function to the cloud only if its risk-adjusted return is larger than a threshold, that is,

$$\frac{E(G)}{\sigma(G)} \geq \alpha. \tag{7}$$

Different applications can use different thresholds according to their properties. For example, some applications are delay tolerant. A small α can be used to raise the expected return. In contrast, for real-time applications, a large α might be preferable.

In the second part, the offload controller adjusts its strategy as the connectivity changes after the offloading decision. For example, the mobile device may lose connectivity before receiving the result. It can choose to wait for the result or execute the function in a local worker. As soon as it is notified of the disconnectivity, the offload controller will re-evaluate the returns and risks of these two options and adjust its strategy according to Formula 7.

7. EVALUATION

In this section, we evaluate how IC-Cloud improves the performance and energy consumption of various mobile applications in different types of mobile environments.

7.1 Methodology

We implemented our prototype of IC-Cloud (i.e., both IC-Cloud server and IC-Cloud client) on the Android OS. The IC-Cloud server runs on Android x86. We use a server with an 8-core 3.4GHz CPU, running VirtualBox 4.1.22, in our lab. The IC-Cloud client runs on a Samsung Galaxy Tab with a 1.0GHz Cortex A8 processor and equipped with both WiFi and 3G connections.

We evaluate IC-Cloud's benefits in three different mobile scenarios:

- *Indoor WiFi*: A student carrying a mobile device randomly walks in our department's building which has good WiFi coverage. WiFi is used for Internet access. In this mobile environment, the mobile user will experience varying signal strength as WiFi APs have limited communication range. However, intermittent connectivity is less frequent.
- *Outdoor WiFi*: A student carrying a mobile device takes a shuttle running on the Georgia Tech campus. WiFi is used for Internet access. In this scenario, the mobile user will experience both varying signal strength and frequent intermittent connectivity.
- *Outdoor 3G*: A student is on his commute between home and school. The mobile device accesses the Internet through an EVDO network of a large US 3G carrier. Compared with WiFi, it has lower bandwith and longer delays but better coverage.

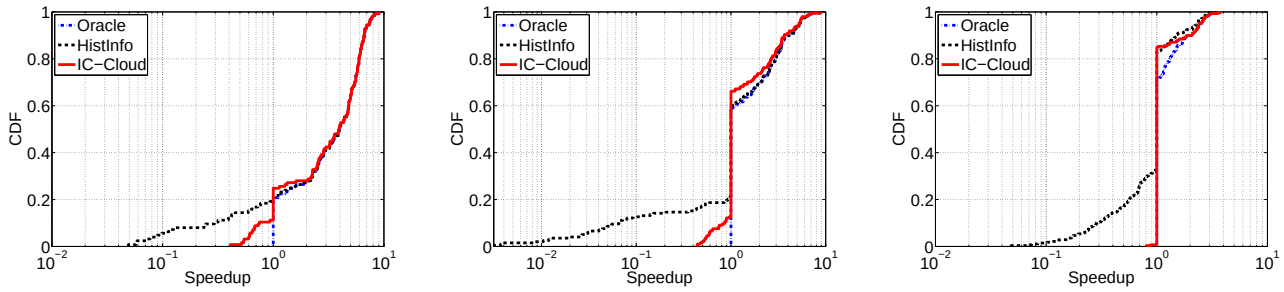
For each scenario, we first measure the network connectivity and construct a database for it. During the experiments, we use those databases as user historical information.

We modified three existing mobile applications to use IC-Cloud for computation offloading:

- FACEDETECT is a face detection application that uses APIs in the Android SDK to detect all the faces in a given picture. We collected a data set of pictures containing faces from Google Image. During the experiments, each time we randomly choose a picture in the data set as input to the application.
- VOICERECOG is an Android port of the speech recognition program PocketSphinx [21]. For simplicity of experiments, we also modified the application to use audio files as input. We created a set of audio files with different lengths in advance. During the experiments, we randomly select a file as the input each time.
- DROIDFISH is an Android port of the Chess engine Stockfish [1] that allows users to set the strength of the AI to play with. Because computation requirement changes with the chosen AI player's strength, the user in our experiments randomly changes the strength before each move.

To evaluate the benefits of IC-Cloud, we use three offloading baselines:

- *LocalExe* executes all offloadable functions locally on the mobile devices. It provides a fundamental baseline to demonstrate the benefit of computation offloading.
- *Oracle* assumes accurate knowledge of all connectivity and execution profile information necessary to make the offloading decision. It represents the upper-bound of offloading benefits.
- *HistInfo* uses user historical information in a simple way that works well if network connectivity is relatively stable. It uses signal strength to estimate the average throughput it can achieve. It also uses past



(a) Indoor WiFi (b) Outdoor WiFi (c) Outdoor 3G

Figure 9: A comparison of IC-Cloud's performance benefits using the FACEDETECT application in different mobile scenarios . In every experiment, a mobile user randomly picks a picture in the repository and detects the faces with the application. The speedups against local execution on the mobile devices are reported. When the speedup is larger than 1, offloading improves application performance.

invocations of a function as a predictor of future invocations [11, 10]. The comparison between HistInfo and IC-Cloud will demonstrate the importance of handling dynamic Internet access quality in mobile environment.

The dynamic mobile environment makes the comparison very hard since each invocation of an offloadable function has different Internet access quality. To achieve fair comparison with those baselines, at runtime we force IC-Cloud to offload every offloadable function and record the information of network connectivity and application states. Then we replay these applications later for each baseline. We are aware that this method may introduce some errors but believe they are negligible.

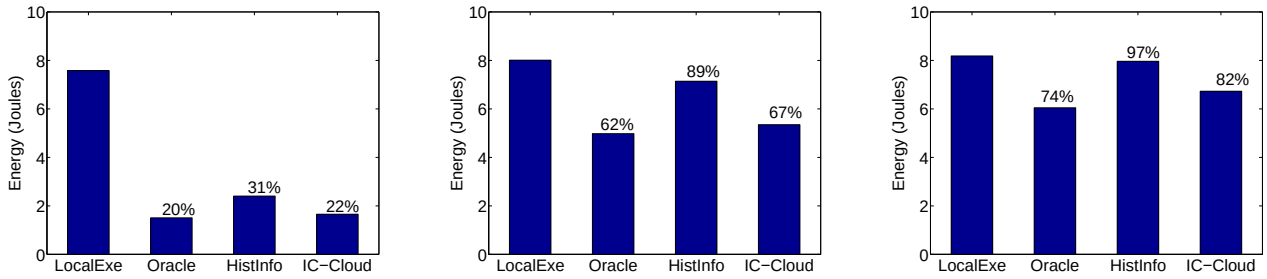
The primary goal of IC-Cloud is to improve the performance of mobile applications. Therefore, we use *speedup*, i.e., $\frac{\text{Execution time}}{\text{Local execution time}}$, as the major metric for evaluation. When speedup is larger than 1, the application benefits from offloading. When speedup is less than 1, it spends more time for execution. We also measure the energy to analyze if IC-Cloud can also reduce energy consumption. We use PowerTutor [32], a power estimation tool for Android, to estimate the power consumption. We will primarily report the energy consumption of CPU and network interfaces because other background energy consumption (e.g., screen) is related to user settings.

7.2 The effect of connectivity scenarios

In the first set of experiments, we evaluate the performance of IC-Cloud in the three different mobile scenarios using the FACEDETECT application. α (see Section 6.2) is set to 0.5 for IC-Cloud. Figure 9 shows the speedup distribution in those experiments. When the speedup is larger than 1, the system outperforms LocalExe. However, when the speedup is smaller than 1, the system causes longer execution time. For example, when speedup is 0.1, IC-Cloud takes 10 times the local execution time. In all these experiments, IC-Cloud performs well and achieves similar performance to Oracle. It also outperforms HistInfo by reducing the number of bad offloading decisions.

We also find some interesting phenomena in these experiments. First, in the scenario of Indoor WiFi where mobile users have good WiFi coverage, offloading computation to the cloud benefits the mobile applications in about 80% of the cases. However, there are still about 20% in which a simple method like HistInfo will increase the execution time as much as 20 times. IC-Cloud reduces the portion of negative cases to about 10% with the smallest speedup at approximately 0.4. In addition, it also enables 75% of the cases to benefit from offloading. IC-Cloud achieves 4.1x overall speedup in this scenario. Second, in the scenario of Outdoor WiFi where intermittent connectivity is common, at most 40% of the cases can benefit from offloading. HistInfo causes the bad offloading to take much more time to execute. In contrast, IC-Cloud still manages to control the smallest speedup in a similar range to Indoor WiFi scenarios. Third, in the scenario of Outdoor 3G, ideally at most 30% of the cases benefit from offloading, while the maximal speedup is only about 4. This is because the 3G has relatively smaller bandwidth and longer delays. In this scenario, IC-Cloud helps about 15% invocations of FACEDETECT benefit from offloading. In addition, it only occasionally made some negative decision, while HistInfo causes more than 30% invocations to take more time to execute.

To demonstrate how IC-Cloud can also save energy for mobile devices, we plot the average energy consumption for every scenario in Figure 10. IC-Cloud only consumes about 22%, 67% and 82% energy of local execution in those three scenarios, respectively. Its energy consumption is also very close to Oracle in all those scenarios. In addition, since IC-Cloud also reduces the execution time of these applications, it may also reduce the background energy consumption (e.g., screen). We also notice that although HistInfo made many bad offloading decisions in the scenario of Outdoor 3G, its average energy consumption is similar to LocalExe. This is because bad offloading decisions correspond to pictures



(a) Indoor WiFi (b) Outdoor WiFi (c) Outdoor 3G

Figure 10: A comparison of IC-Cloud's energy consumption using the FACEDETECT application in three different mobile environments. We primarily report the energy consumption of CPU and network interfaces.

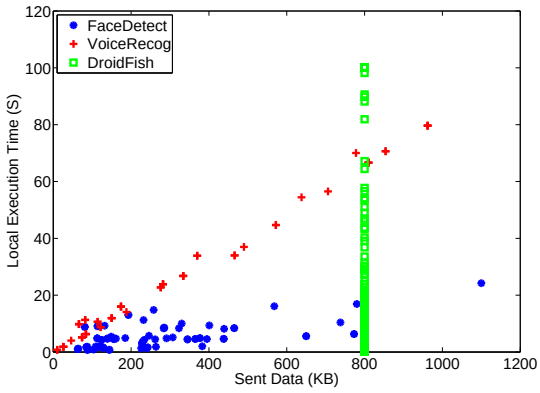


Figure 11: The local execution time vs. the size of uploaded data

with small local execution times. Although for each case it took much more energy to offload, the total extra energy is relatively small and is compensated by the energy saving of offloading other larger tasks in the experiments.

7.3 Results for different applications

The gain from computation offloading is normally counterbalanced by the communication cost. Different applications usually have different execution times and different amount of data exchanged between the mobile device and the cloud. In this subsection, we evaluate IC-Cloud with different applications of different properties on computation and communication cost. Figure 11 plots the local execution time of the offloadable functions and the corresponding data to be sent to the cloud. We can see that local execution time of VOICERECOG is almost proportional to the data size, while DROIDFISH has constant data size.

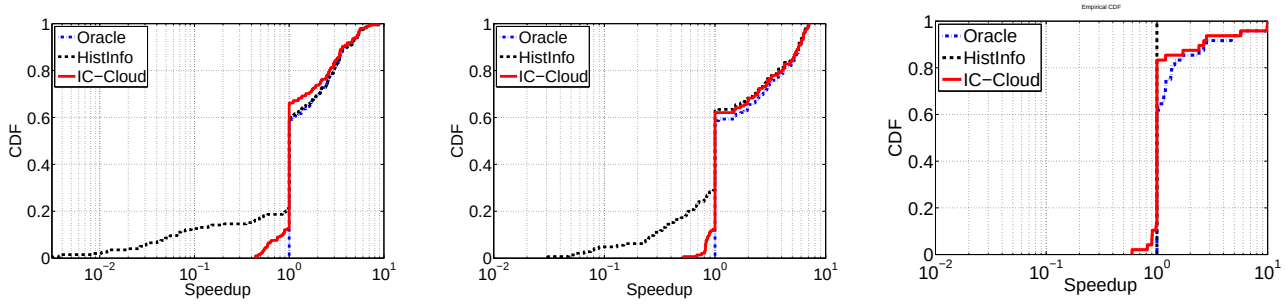
To demonstrate how these application properties impact computation offloading, we compare the performance of different applications using IC-Cloud in the scenario of Outdoor WiFi. The results are plotted in Figure 12.

IC-Cloud performs well in all these experiments. In FACEDETECT and VOICERECOG, IC-Cloud helps more than 35% of the function invocations benefit from offloading. Meanwhile, it limits the portion of bad decisions cases to be about 10% with small extra execution time. In contrast, HistInfo makes many more bad offloading decisions and causes these invocations to last much longer. Compared with FACEDETECT app, HistInfo made about 30% invocations execute longer than LocalExe. This is because HistInfo's application prediction method can easily overestimate the local execution time and, thus, mistakenly decide to offload them. In contrast, our application prediction method helps IC-Cloud obtain accurate prediction and avoid the unnecessary risk in computation offloading.

The behavior of DROIDFISH is quite different from those of FACEDETECT and VOICERECOG. Even Oracle can only help about 15% of those invocations achieve more than 2x speedup. This is because the data uploaded to the cloud is so large that if the computation gain is small it cannot compensate for the communication cost. As in our experiments only a small portion of invocations have long local execution time, the overall performance improvement is small. However, for those invocations with long local execution time, DROIDFISH can still benefit from offloading. We notice that HistInfo always chooses to execute locally because it underestimates the computation gain using previous invocations. IC-Cloud helps about 20% of the invocations improve their performance and about 15% achieve 2x speedup. Compared with Oracle, IC-Cloud does not help computations that can only achieve small performance improvement because it tries to control the risk of offloading. As a result, only a small portion of invocations have longer execution time when using IC-Cloud.

7.4 The performance of system components

IC-Cloud has three key components, including connectivity predictor, application predictor and offloading controller. In this subsection, we evaluate the performance of various system components to show how these components help IC-Cloud achieve good performance in computation offloading. As the results of connectivity predictor have already been shown in Section 4, we will focus on the latter two compo-



(a) FACEDETECT (b) VOICERECOG (c) DROIDFISH

Figure 12: A comparison of IC-Cloud's performance with three different applications. All experiments are conducted in the scenario of Outdoor WiFi.

	SLOC	# features				# inputs	
		loop	retn.	param	chosen	train	test
FACEDETECT	225	0	2	3	2	31	32
VOICERECOG	772	1	3	14	1	34	35
DROIDFISH (server)	23,662	2	1	18	2	246	246
DROIDFISH (tablet)		2	1	18	3	144	145

Table 1: Statistics of program features and inputs used for execution prediction.

nents here.

7.4.1 *Application performance prediction*

We run the instrumented application with a number of test cases on the Samsung Galaxy Tab and the server to build prediction models. We run 63 test cases for FACEDETECT app, 69 for VOICERECOG app, and 289 for DROIDFISH app on Samsung and 492 on server. When machine learning is working, we set the portion to be 0.5, so that half of the test cases would be used to generate formulae, and the remaining half to test.

Table 1 shows the statistics of program features and inputs for execution prediction. For FACEDETECT app, the offloadable function detectFaces() is used to detect the faces in the current picture. Before it is called, the Profiler (see Section 5.1) discovers 5 features available: 1 irrelevant parameter feature, 1 parameter and 1 return value both point to the width of decoded picture, 1 parameter and 1 return value both point to the height of decoded picture. As shown in Table 2, machine learning always chooses 1 feature (f_1) for the width of picture and 1 feature (f_2) for the height of picture to generate the formula.

For VOICERECOG app, the offloadable function audioRecog() is used to recognize the words in the current audio file. Before it is called, the Profiler discovers 18 features available: only 1 of them are related to the runtime of the function, and they both point to the length of the audio file. As shown in Table 2, machine learning chooses 1 of them to generate the formula.

For DROIDFISH app, we only instrument the class that contains the offloadable function, otherwise the app runs

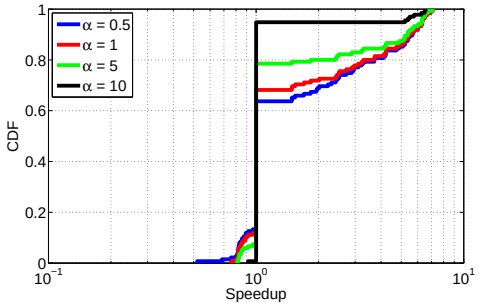


Figure 14: The tradeoff between risk and return. We use different values of α for VOICERECOG in the scenario of Outdoor WiFi.

extremely slowly under full instrumentation. The function whose running time requires estimation is the function iteractiveDeepening(), which dominates the running time. The function iteractiveDeepening() is used to find the next move based on the user's current move and game state by searching a series of moves specified by a maximum search depth. Before the function iteractiveDeepening(), the Profiler discovers 21 features available: only 6 of them are related to the running time of the function, and 4 of them are related to strength (maximum search depth) and the others are the size of chess move history list. As shown in Table 2, machine learning chooses 2 or 3 of them to generate a formula, in which f_1 and f_2 are related to the strength, and f_3 is the size of the chess move history list.

The experiment results are shown in Table 2 and Figure 13. Both FACEDETECT and VOICERECOG have good prediction performance with average error rate less than 10%. However, for the DROIDFISH app, the estimated points are spread far away from the 45 degree line, especially when actual running time is very large. The reason is that the search procedure is bounded by minimum and maximum time limits. When the search procedure hits the maximum time limit, the running time does not depend on the feature variables any longer. So the ideal formula should be in the form of a piecewise function, which is too sophisticated for machine

	performance model		average % prediction error	
	server	tablet	server	tablet
FACEDETECT	$0.2 + 2.4f_1 + 0.4f_2 + 7.4f_1f_2$	$0.2 + 2.1f_1 + 0.6f_2 + 6.8f_1f_2$	6.40	5.71
VOICERECOG	$0.1 + 10.2f_1 - 0.3f_1^2$	$10.2f_1$	5.99	7.11
DROIDFISH	$0.005 + 0.815f_1^2f_2 - 0.715f_2^2$ $+0.093f_2 + 0.085f_1f_2^2$	$0.023 - 1.658f_1^2 + 3.044f_1^2f_2$ $+0.243f_2 + 0.037f_3^2$	59.53	53.16

Table 2: Execution prediction models and error rates using IC-Cloud.

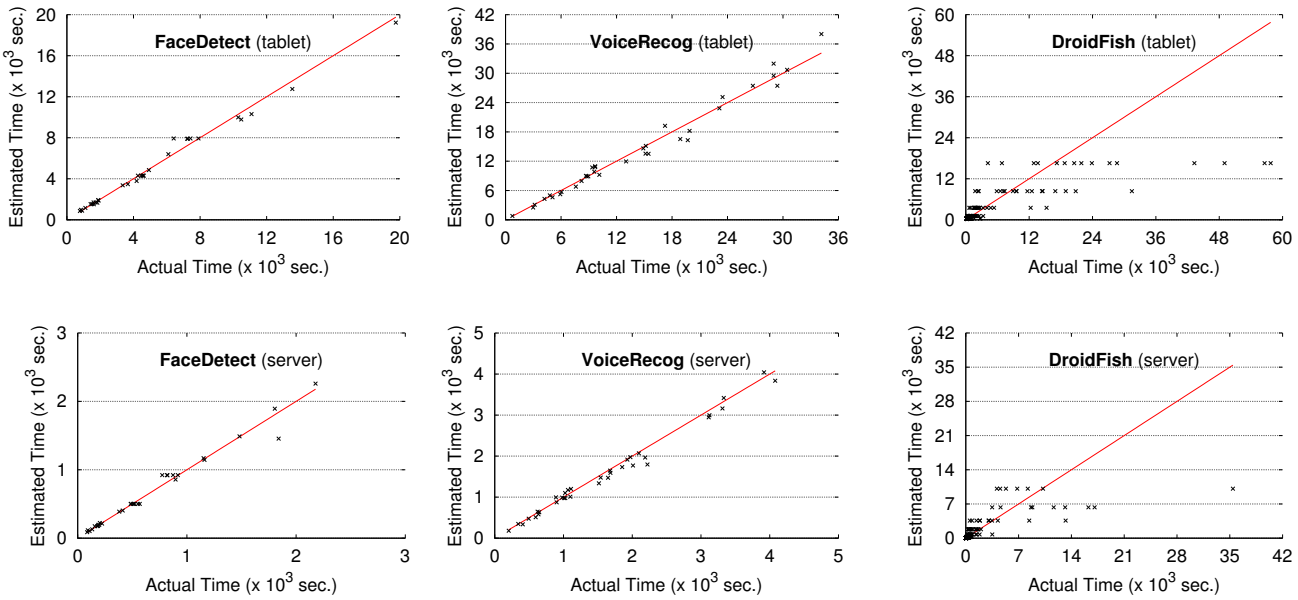


Figure 13: Accuracy of the execution prediction performed by IC-Cloud.

learning to derive. Moreover, the player's skills will affect the search depth, which in turn perturbs the running time of the search procedure.

7.4.2 The return-risk tradeoff

IC-Cloud enables applications to control the risk of offloading by setting the value of α. An application sensitive to extra delays can use large α value, while a small α value will result in higher expected return. To show how α impacts the return-risk tradeoff, we apply various values of α to the VOICERECOG application in the scenario of Outdoor WiFi. Figure 14 plots the results.

When the value of α increases from 0.5 to 10, the portion of invocations with speedup less than 1 decreases from about 10% to almost 0%. Meanwhile the portion of invocations that can benefit from offloading also drops from about 35% to 5%. It will be important to find a proper tradeoff between return and risk a question we relegate to future research.

8. CONCLUSION AND FUTURE WORK

In this paper, we proposed IC-Cloud, a system for computation offloading in mobile environment where the Internet access to remote computation resources is of highly varying quality and often intermittent. IC-Cloud uses three key techniques to overcome the uncertainties in this environment, including lightweight connectivity prediction, lightweight execution prediction and usage of these predictions in a risk controlled manner to make offloading decisions. We have implemented IC-Cloud on Android. Our evaluation explored a large space of possibilities by testing in three different mobile connectivity scenarios and three applications with differing computation and data I/O profiles. The experimental results show that IC-Cloud can enable effective computation offloading in a variety of mobile environments.

There are a number of future issues to consider. These include a further exploration of the risk-return tradeoff and understanding how to optimize it in particular settings. We also would like to explore how our system can be used within the context of paid cloud services where remote execution may have a monetary cost especially that in our context the remote cloud can be idle-waiting (and incurring cost) waiting for reconnection to the mobile device even after a computation is completed.

9. REFERENCES

[1] Droidfish. `http://web.comhem.se/petero2home/droidfish`.

[2] Siri. http://www.apple.com/ios/siri.

[3] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *ACM SIGOPS European workshop*, 2002.

[4] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb. Simplifying cyber foraging for mobile devices. In *ACM MobiSys*, 2007.

[5] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, MobiSys '03, pages 273–286, New York, NY, USA, 2003. ACM.

[6] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3g using wifi. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 209–222, New York, NY, USA, 2010. ACM.

[7] J. Berk and P. DeMarzo. *Corporate finance*. Addison-Wesley, 2007.

[8] D. Bertsekas and J. Tsitsiklis. *Introduction to probability*, volume 1. Athena Scientific Nashua, NH, 2002.

[9] B. Chun, L. Huang, S. Lee, P. Maniatis, and M. Naik. Mantis: Predicting system performance through program analysis and modeling. *Computing Research Repository (CoRR)*, 2010.

[10] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the 6th European Conference on Computer Systems (EuroSys'11)*, pages 301–314, 2011.

[11] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *ACM MobiSys*, 2010.

[12] P. Deshpande, X. Hou, and S. R. Das. Performance comparison of 3g and metro-scale wifi for vehicular network access. In *Proc. ACM IMC*, 2010.

[13] P. Deshpande, A. Kashyap, C. Sung, and S. R. Das. Predictive methods for improved vehicular wifi access. In *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, pages 263–276, New York, NY, USA, 2009. ACM.

[14] J. Flinn. Cyber foraging: Bridging mobile and cloud computing. *Synthesis Lectures on Mobile and Pervasive Computing*, 7(2):1–103, 2012.

[15] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. Jordan, and D. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. pages 592–603, 2009.

[16] A. Gember, C. Dragga, and A. Akella. Ecos: leveraging software-defined networks to support mobile application offloading. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, ANCS '12, pages 199–210, New York, NY, USA, 2012. ACM.

[17] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen. Comet: code offload by migrating execution transparently. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 93–106, Berkeley, CA, USA, 2012. USENIX Association.

[18] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic. Adaptive offloading inference for delivering applications in pervasive computing environments. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, PERCOM '03, pages 107–, Washington, DC, USA, 2003. IEEE Computer Society.

[19] C. Gupta, A. Mehta, and U. Dayal. PQR: Predicting query execution times for autonomous workload management. pages 13–22, 2008.

[20] L. Huang, J. Jia, B. Yu, B.-G. Chun, P. Maniatis, and M. Naik. Predicting Execution Time of Computer Programs Using Sparse Polynomial Regression. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[21] D. Huggins-Daines, M. Kumar, A. Chan, A. Black, M. Ravishankar, and A. Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 1, pages I–I. IEEE, 2006.

[22] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *IEEE Infocom*, 2012.

[23] R. Mahajan, J. Zahorjan, and B. Zill. Understanding wifi-based connectivity from moving vehicles. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 321–326, New York, NY, USA, 2007. ACM.

[24] A. J. Nicholson and B. D. Noble. Breadcrumbs: forecasting mobile connectivity. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, MobiCom '08, pages 46–57, New York, NY, USA, 2008. ACM.

[25] J. Porras, O. Riva, and M. Kristensen. Dynamic resource management and cyber foraging. *Middleware for Network Eccentric and Mobile Applications*, 1:349, 2009.

[26] V. S. P. L. E. G. Raja Vallee-Rai, Laurie Hendren and P. Co. Soot - a java optimization framework. In *Proceedings of CASCON 1999*, pages 125–135, 1999.

[27] M. Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*,

8(4):10–17, 2001.

[28] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 2009.

[29] C. Shi, M. H. Ammar, E. W. Zegura, and M. Naik. Computing in cirrus clouds: the challenge of intermittent connectivity. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, MCC '12, pages 23–28, New York, NY, USA, 2012. ACM.

[30] C. Shi, V. Lakafosis, M. Ammar, and E. Zegura. Serendipity: Enabling remote computing among intermittently connected mobile devices. In *ACM MobiHoc*, 2012.

[31] J. a. P. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance*, WICSA 3, pages 29–43, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.

[32] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, CODES/ISSS '10, pages 105–114, New York, NY, USA, 2010. ACM.

APPENDIX

In this appendix we describe how to compute the offloading gain G of a single function at time t. Depending on the connectivity at time t, T_{ws} and T_s have different distributions. In the case that the mobile device connects to the cloud at time t, $T_{ws} = 0$; $T_s = \frac{d_s}{b_u(t)}$, where d_s is the data size, and $b_u(t)$ is the upload bandwith at time t. d_s is available at time t, while $b_u(t)$ can be estimated using the current signal strength as described in Section 4.3. Otherwise, $T_{ws} = R_{D,t}$, which can be obtained according to Equation 1 and 2; $T_s = \frac{d_s}{b_u^*}$, where b_u^* is the overall upload bandwidth of the entire trace.

The value of T_{wr} depends on whether the mobile device still connects to the cloud when the cloud finishes execution at time $t+T_{ws}+T_s+T_c$. If connected, $T_{wr} = 0$. Otherwise,

$$T_{wr} = \begin{cases} D - R_{C,t} + T_s + T_c, & \text{if connected at time } t \\ D - C + T_s + T_c, & \text{otherwise} \end{cases} \quad (8)$$

where C and D are contact duration and inter-contact duration, respectively.

The value of T_r also depends on the connectivity at time $t' = t + T_{ws} + T_s + T_c$. If connected, $T_r = \frac{d_r}{b_d(t')}$, where d_r is the result size, and $b_d(t')$ is the download bandwidth. $b_d(t')$can be estimated according to Formula 5. Otherwise $T_r = \frac{d_r}{b_d^*}$, where b_d^* is the overall download bandwidth.

According to the above analysis, T_{wr} is directly related to T_s and T_c. T_r is indirectly related to T_s and T_c as $T_s + T_c$ may impact the distribution of signal strength which impacts T_r. However, this correlation is small and, thus, be ignored in the implementation for simplicity. Other variables are independent of each other. Therefore, the variance of offloading gain can be computed using

$$\begin{aligned} \sigma^2(G) &= \sigma^2(T_{ws}) + \sigma^2(T_s) + \sigma^2(T_c) + \sigma^2(T_{wr}) \\ &\quad +\sigma^2(T_r) + \sigma^2(T_l) + 2\sigma(T_s + T_c, T_{wr}) \end{aligned} \quad (9)$$